**UML For Class Diagram and Sequence Diagram | Design Diagrams**

**Introduction:** This section is about UML (class Diagram).

**UML** (Unified Modeling Language):- UML is a **standard way to visualise the software design.**

**UML** is a Unified Modeling Language **that uses notations and diagrams** so that it is **easy to communicate between developers and architects.**

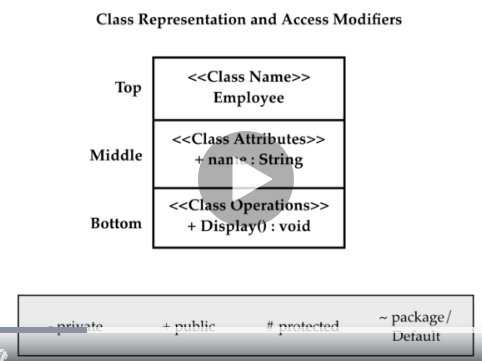
UML makes everybody to speak the same language to reduce the communication gap and for better understanding.

We can say if UML was not there then everybody would have created the different type of diagram for the same problem and it might be very hard to communicate among the community of developers and architects.

So, here we will use UML sequence and class diagrams heavily in design pattern. That is why it’s necessary to know about the UML diagram before going to that.

**Class Diagram**

What is a **class diagram**? – A class diagrams are the main building block in object-oriented modeling. They are used to show the different object in the system, their attributes, operations and relationship among them.



Above if you see there is a class called **Employee** which is depicted. Classes in class diagram are represented by boxes that are partitioned into three sub-partitions. The **TOP** partition contain the **name of the class**, The **MIDDLE** part contains the **class attributes** and the **BOTTOM** partition contain / shows the possible operations that are associated with the class.

While declaring the attributes and operations we have to take care of the **Access-Modifiers.** You can see the access modifiers and the symbol used for those modifiers **– private, + public, # protected, ~ Package / Default.**

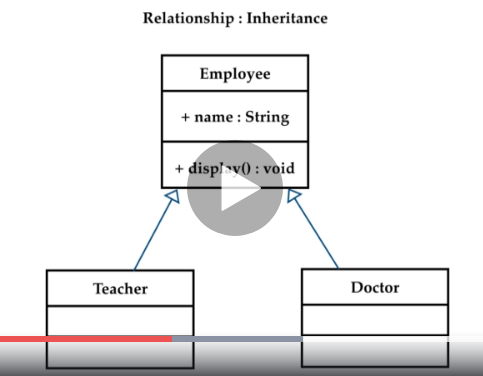
In above image if you see in the **MIDDLE PART** we have name attributes with + symbol which denotes public and similarly in **BOTTOM PART +** Display() as public method.

So, after designing, all these classes are interrelated to each other in specific way so for these relationship in class diagram. It includes different type of logical connection.

Like first one is inheritance – that the child is related to the parent class. So, to show the inheritance in UML diagram there is a solid line from child class to the parent class which is drawn using unfiled arrow head.

Let’s create an **Employee** class.

Let’s create a two more classes like **Teacher** and **Doctor** and then now create the relationship between these three classes (Employee, Doctor and Teacher)

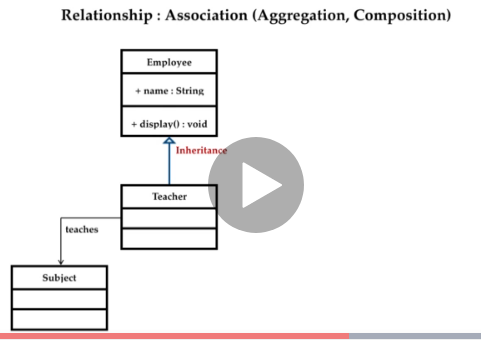


Now the Teacher and Doctor classes are the child / sub-classes of the Employee class and Employee class is the parent / super class.

So, in all these scenarios we also have abstraction.

Employee is abstract class not really but, in our system, anytime we can instantiate one of these two classes. Either Teacher OR Doctor class. And we don not need to instantiate Employee class directly because we are overriding the functionalities in Teacher OR Doctor class and we can use that functionalities using those classes. He is a Teacher, OR He is a Doctor accordingly.

Then there is another relationship comes which is **Association(Aggregation, Composition)**

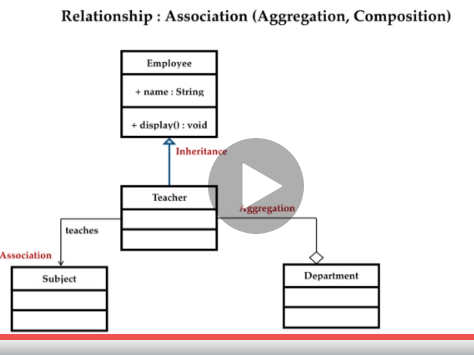
****

**Association:-** That if we add the class **subject** that connects with a line open – ended arrow and then you can write on it “teaches” to focus what it does. So, it’s like Teacher teaches subject. It’s a **HAS-A relation**. Like A Teacher has a subject.

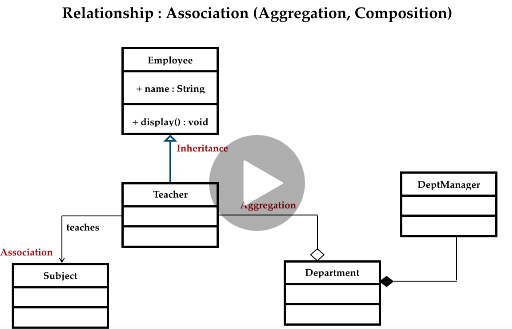
**Association is of two type: Aggregation and Composition.** Like we are adding one more class **Department** – which denoted the group of teachers. So, here **Department** class got an aggregation relationship between Techer and Department. Any of the school teacher could be a part of the department but they don’t necessarily have to be. A Teacher could leave the department at any point and still exist in its own.

This type of relationship where part can exist outside the whole is the aggregation.

For this we use unfilled diamond head arrow towards the containing object.

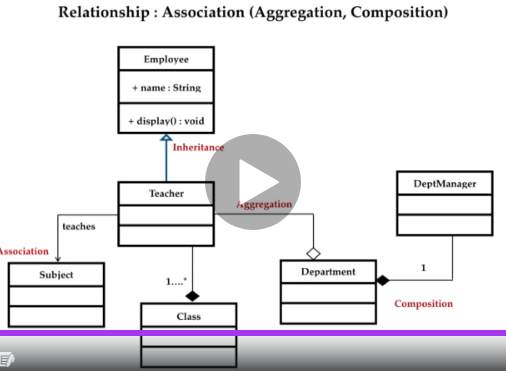


Now let’s see the **composition**: - Below we have class called **DeptManager**, So, each department has **DeptManager** so, if any of the department closed then department manager will also not exist, and it’s called **Composition.** Where contained object would not be able to exist without its container or containing object. For this we use arrow 🡪 with diamond head from the contained object towards the containing object. So, these were a **HAS-A** relationship which we implemented using the class diagram.



We have some another concept like **multiplicity:-** which allow us to set numerical constraints on a created relationship.

Let’s say we want to specify that a **Department** is going to have one department manager(**DeptManager**) only then we can write one(**1**) on the line of **Composition.** But we can have at least one teacher per class. So, for one or many we write one followed by dots and astric like **1…\***

****

For one to one relationship we write **1** and

For one to many relationships we write **1…\***

**Sequence Diagram:-** Sequence diagram is a type of UML diagram, that denotes how objects and classes within the code interacts with each other at runtime.

Particularly, they show sequence of events.

Sequence diagrams are used for **documents**, **processing** or to understand the requirements of a program or application.

Before we discuss the sequence diagram, we need to show a sample example or topic on which we will design our sequence diagram.

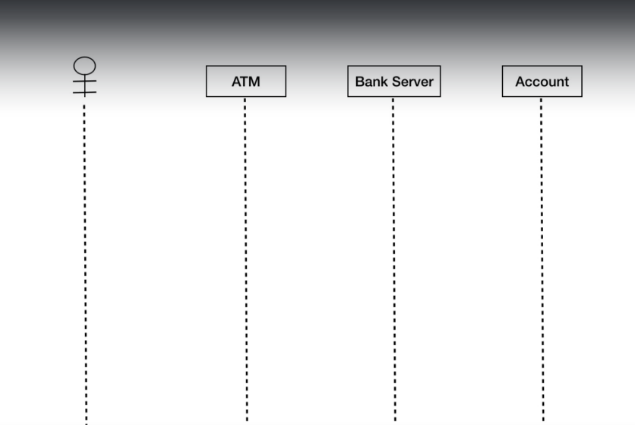
Example: Let’s make a diagram to ATM interactions.

First of all, we need to think of all the parts that are required to make up what is needed.

We are taking the example of ATM so, what are the parts of atm transaction process.

The process will be like a person goes to ATM, pinch the card, then ATM verify the cards and then if valid then ok otherwise the card gets rejected. If valid then ATM ask for the pin, again the same process – valid pin or not valid pin, it will interact with that person accordingly.

If the PIN is valid then ATM will ask the amount withdrawn and then it will connect to the bank server again. Then deducts the amount from the account and then will give us the requested money and then eject the card.



In this process the parts were ATM, Bank Server, and Account which hare represented as object.

And the person is called as Actor and actor is always the outside of scope of the system in sequence diagram. 🡪 Always Remember.

Since in sequence diagram the actor is represented as a stick figure and object with rectangle as you can see in above diagram.

Actor denotes the person who is requesting the money from ATM. Then we have three object ATM, Bank Server, and Account. Always remember objects are always place in sequential order from left to right in your story or process.

Then we need to add the lifeline. Lifelines are the vertical dash which shows the existence of object or actor overtime. So, we draw out the lifeline for each four of the objects. So, the base structure is now ready and now we can start the interaction between the actor and other objects. We can think of all the steps which we have discussed earlier. The first thing we do is **:-**

**Inserting a card into the ATM. 🡪** so, it will be the first message andthen

**Connect an actor to the ATM by one arrow solid line[🡪].** After this

**ATM will interact with Bank Server to validate (verify card) the card and then will sends the message back with dashed strip line (<- - - - Card OK)** then

**ATM will ask the information from user like ATM PIN (Drag an arrow solid line – Enter PIN)**

**Why solid line? 🡪 because it’s not a return message / response /reply but it’s an instruction.**

**SOLID LINE \_\_\_\_\_\_\_ for instruction.**

**Stripped LINE: - - - - - - - - - for returned message / response / reply**

But what happened if the card is invalid! So, here comes two condition. Either Valid OR Invalid so **IF-ELSE** condition.

In UML the **IF-ELSE** condition is represented by **alternative frame 🡪**

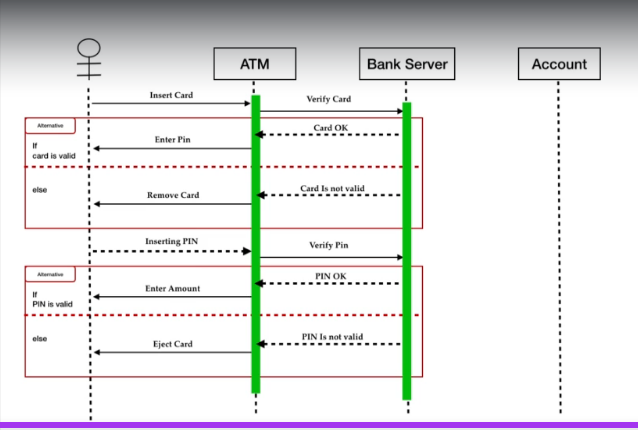
Like given in above diagram.

Which is used to choose **one** between **two** or more message sequences.

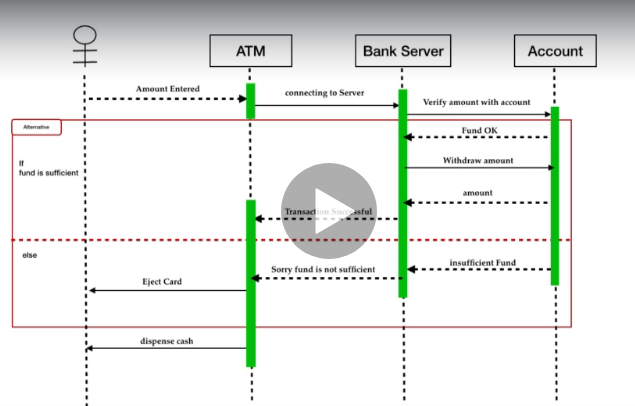
Set the alternative box and put it around the messages upper portion of the box denotes **IF** condition and lower will denotes the **ELSE** block.

So, if the card is valid then ATM will ask for the PIN user will respond to the ATM. SO, this time we dragged a line Actor to ATM and change it from **SOLID** to **DASHED.**

**Again, ATM** will verify the PIN with bank server same condition arises like card VALID / INVALID so alternative frame and perform the operation. If valid then ATM will request the amount ELSE will says to eject the card.

Example: 

So, after this we will continue to map the remaining interaction by creating return messages until transaction complete and actor will receive the cash.



Now activation boxes will come into picture. Activation boxes assures when how long and object is performing the process. That means it shows when objects were active and when idle.

Now the user will input the amount, The ATM will start the transaction with bank server now the server will ask to bank account that the fund is sufficient to cover the transaction. If YES, then account send the return message. That funs is ok server will withdraw the amount. Then says transaction is successful dispense the cash if not then again alternative frame will come into picture. ELSE insufficient funds then transaction unsuccessful and eject the card. So, here we have completed transaction process again activation boxes will come into picture.

Activation boxes will show when and how long an object is performing the process. That means it shows when objects are active or when idle. We just add activation boxes on the ATM bank server and bank account.

As discussed earlier that external object and actor don’t have any activation boxes.

Because it’s out of the scope.

This sequence diagram is a simple, but this can get even more complex that if you are diagraming code then object might be labelled as classes and messages might be as methods used in the codes.

It all depends on what all we need to do in diagram and how complex your need it to be. So this was all for sequence diagram.